Counting Protein Structures by DFS with Dynamic Decomposition

Sebastian Will and Martin Mann

Chair for Bioinformatics, Institute of Computer Science, Albert-Ludwigs-University
Georges-Koehler-Allee, Geb. 106, D-79110 Freiburg, Germany
{will,mmann}@informatik.uni-freiburg.de

Abstract. We introduce depth-first search with dynamic decomposition for counting the solutions of a binary CSP completely. In particular, we use the method for computing the number of minimal energy structures for model proteins.

1 Introduction

The number of minimal energy structures of model proteins is an important measure, which is strongly related to protein stability. The enumeration of optimal and suboptimal structures has applications in the study of protein evolution and kinetics [12, 20, 26, 25]. The prediction of protein structures in simplified protein models is a complex, NP-complete [6] combinatorial optimization problem that received lots of interest in the past, e.g. [16, 27]. Importantly for our work here, it can be successfully modeled as Constraint Satisfaction Problem (CSP) [2, 4].

Recently, counting solutions of a CSP and related problems gained a lot of interest over considering only satisfiability [1, 9, 17, 22]. This is partly due to the increased complexity of counting compared to deciding on satisfiability [19]. Standard solving methods in constraint programming like Depth-First Search (DFS) combined with constraint propagation are therefore well suited for determining one solution, but leave room for saving redundant work when counting all solutions. Here, we present a method that is especially tailored for this case. Applied to the CSP formulation of structure prediction, it improves exhaustive counting and enumeration of optimal protein structures.

Basically, our new method *dynamically* decomposes the constraint (sub-)problems that emerge during the search into independent partial problems along connected components of the problem's associated constraint graph. Separate counting in the partial problems still allows to infer the number of solutions of the complete problem.

Instead of *statically* exploiting only properties of the initial constraint graph, dynamic strategies analyze the emerging constraint graphs during the search and employ their features. We believe this is a major advantage in many constraint problems. In particular, if the initial constraint network is very dense (as in our structure prediction problem), static methods don't make an impact.

Decomposing into connected components and, more generally, utilizing the special structure of the constraint graph is discussed already for a long time. In the beginning, [13] proposed statically decomposing a CSP and solving the partial problems independently. As a more recent example, [9] introduced AND/OR search for solution counting, again this approach relies on static analysis of the constraint graph. To our knowledge, dynamic decomposition was discussed more thoroughly only for very special cases. [17] showed that adding this idea to counting models of 3-SAT by a Davis-Putnam procedure [8] results in a very successful new strategy. Similar ideas are discussed for SAT-solvers in [7].

As our main contribution, we demonstrate that the ideas of employing the graph structure dynamically are applicable to binary CSPs, even including certain global constraints, and are useful for constraint programming. In particular, this allows us to use the strategy in the complex problem of protein structure counting. Furthermore, we discuss several ideas going beyond previous approaches. For example, dynamic decomposition can yield a more compact representation of the solution space. We discuss how analyzing the constraint graph can further improve counting and how search strategies can be tailored in order to maximize the benefits from our strategy.

2 Dynamic Decomposition

Definitions A *Constraint Satisfaction Problem (CSP)* is a triple $(\mathcal{X},\mathcal{D},\mathcal{C})$ of variables $\mathcal{X} = X_1,\ldots,X_n$, associated domains $\mathcal{D} = D_1,\ldots,D_n$, i.e. finite sets of values, and a finite set of constraints $\mathcal{C}$ on the variables in $\mathcal{X}$. A *solution* of the CSP is an assignment of each variable in $\mathcal{X}$ to one value in its associated domain. A variable X_i is *determined by* $\mathcal{D}$, iff its associated domain D_i in $\mathcal{D}$ is singleton. We call $(\mathcal{X},\mathcal{D},\mathcal{C})$ *solved*, iff each variable in $\mathcal{X}$ is determined by $\mathcal{D}$. The CSP is *failed*, iff at least one of variables in $\mathcal{X}$ has an empty domain. A *subproblem of a CSP* $(\mathcal{X},\mathcal{D},\mathcal{C})$ is a CSP $(\mathcal{X},\mathcal{D},\mathcal{C}')$, where $\mathcal{C} \subseteq \mathcal{C}'$. A CSP $(\hat{\mathcal{X}},\hat{\mathcal{D}},\hat{\mathcal{C}})$ is called *partial problem of* $(\mathcal{X},\mathcal{D},\mathcal{C})$, where $\hat{\mathcal{X}} \subseteq \mathcal{X}$ and $\hat{\mathcal{D}}$ and $\hat{\mathcal{C}}$ are restrictions of $\mathcal{D}$ and $\mathcal{C}$ to $\hat{\mathcal{X}}$, respectively. We call a CSP *n-ary*, iff each of its constraints is at most n-ary. For a constraint c, we denote by $X(c)$ the *set of variables of* c. The *constraint graph* of a binary CSP $(\mathcal{X},\mathcal{D},\mathcal{C})$ is the undirected graph (V,E) defined by $V = \mathcal{X}$ and $E = \{(X_i,X_j) \in \mathcal{X}^2 | c \in \mathcal{C}, \{X_i,X_j\} \subseteq X(c), X_i \neq X_j\}$. Two partial CSPs $(\hat{\mathcal{X}},\hat{\mathcal{D}},\hat{\mathcal{C}})$ and $(\hat{\mathcal{X}}',\hat{\mathcal{D}}',\hat{\mathcal{C}}')$ of $(\mathcal{X},\mathcal{D},\mathcal{C})$ are *independent*, iff $\hat{\mathcal{X}}$ and $\hat{\mathcal{X}}'$ don't share variables and there is no constraint c in $\mathcal{C}$, where $X(c)$ shares elements with $\hat{\mathcal{X}}$ and $\hat{\mathcal{X}}'$.

Counting DFS The usual approach to counting solutions of a CSP is by DFS in combination with constrained propagation. As preparation to our approach, we present a recursive formulation, which we temporarily call Counting Depth-First Search (CDFS).

```
1: function CDFS($\mathcal{X},\mathcal{D},\mathcal{C}$)
2:     $(\mathcal{D}',\mathcal{C}') \leftarrow$ PROPAGATE($\mathcal{X},\mathcal{D},\mathcal{C}$)
3:     if ISFAILED($\mathcal{X},\mathcal{D}',\mathcal{C}'$) then return 0
4:     else if ISSOLVED($\mathcal{X},\mathcal{D}'$) then return 1
5:     else  $c \leftarrow$ SELECT($\mathcal{X},\mathcal{D}'$)
6:         return CDFS($\mathcal{X},\mathcal{D}',\mathcal{C}'\cup\{c\}$) + CDFS($\mathcal{X},\mathcal{D}',\mathcal{C}'\cup\{\neg c\}$)
7:     end if
8: end function
```

In our formulation, $CDFS(\mathcal{X},\mathcal{D},\mathcal{C})$ yields the number of solutions to $(\mathcal{X},\mathcal{D},\mathcal{C})$. Note that the function performs full propagation of constraints to the domains (also, entailed constraints of $\mathcal{C}$ are removed in $\mathcal{C}'$) in line 2. The tests for failure and determination by the propagated domains $\mathcal{D}'$ are in line 3 and 4. Line 6 allows the algorithm an arbitrary branching selection; often one selects a variable X_i from $\mathcal{X}$ and a value $d \in \mathcal{D}_i$ and enumerates by $c = (X_i \equiv d)$. Finally, the solution count of each subproblem adds to the total number of solutions in line 7.

We provide an example CSP and a corresponding search tree for CDFS solution counting in Figure 1. Each node corresponds to a subproblem of the initial problem given in the root and is visualized as a constraint graph.

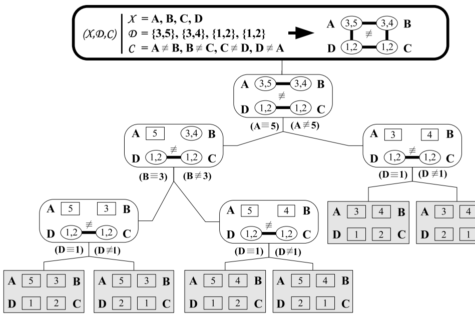

Fig. 1. DFS search tree traversed by CDFS.

Dynamically Decomposing DFS Even in the minimal example of Figure 1, the main problem of CDFS is visible. The partial problem on variables C and D is solved redundantly in each of the search branches. This could be saved due to the independence of the two partial problems on variables A and B and variables C and D. Our new method *Decomposing Depth-First Search (DDFS)* avoids such unnecessary work.

```
 1: function DDFS($\mathcal{X}, \mathcal{D}, \mathcal{C}$)
 2:     $(\mathcal{D}', \mathcal{C}') \leftarrow$ PROPAGATE($\mathcal{X}, \mathcal{D}, \mathcal{C}$)
 3:     if ISFAILED($\mathcal{X}, \mathcal{D}', \mathcal{C}'$) then return 0
 4:     else if ISSOLVED($\mathcal{X}, \mathcal{D}'$) then return 1
 5:     else  $s \leftarrow 1$                                    ▷ initialize counter
 6:         $\mathfrak{D} \leftarrow$ DECOMPOSE($\mathcal{X}, \mathcal{D}', \mathcal{C}'$)
 7:         for all $(\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}}) \in \mathfrak{D}$ do
 8:             $c \leftarrow$ SELECT($\hat{\mathcal{X}}, \hat{\mathcal{D}}$)
 9:             $s = s \cdot \big( \mathrm{DDFS}(\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}} \cup \{c\}) + \mathrm{DDFS}(\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}} \cup \{\neg c\}) \big)$
10:         end for
11:         return $s$
12:     end if
13: end function
```

The code differs from CDFS only in lines 5 to 11, which correspond to the decomposition into independent partial problems. In line 6, we completely decompose the propagated CSP $(\mathcal{X}, \mathcal{D}', \mathcal{C}')$ into its pairwise independent partial problems.

Note that the independent partial problems correspond to the connected components in the constraint graph. Consequently, our decomposition can be computed in linear time by depth first traversal of the graph. As a technicality, we fuse all solved partial problems to an (arbitrary) unsolved partial problem. In consequence, all remaining problems in $\mathfrak{D}$ are unsolved. In line 11, s is

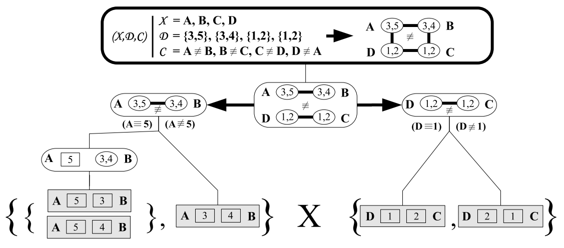
(X,D,C)
X = A, B, C, D
D = {3,5}, {3,4}, {1,2}, {1,2}
C = A ≠ B, B ≠ C, C ≠ D, D ≠ A
(A ≡ 5)
(A ≠ 5)
(D ≡ 1)
(D ≠ 1)
X

Fig. 2. Search tree traversed by DDFS.

the product of the solution counts of all CSPs in $\mathfrak{D}$. Since $\mathfrak{D}$ is a complete decomposition of $(\mathcal{X}, \mathcal{D}', \mathcal{C}')$ into pairwise independent partial problems, s equals the number of solutions for $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

Using this extension the CSP in Figure 1 can be solved as given in Figure 2 with DDFS avoiding the redundant work. With only one decompositions and two branchings instead of five branchings the overall solution number can be determined.

Note that a simple modification of the counting algorithm yields a nice enumeration strategy. Instead of adding and multiplying solution counts, we can build up a tree-like compact representation of the solution space. Examples are given at the bottom of Figure 2 and later in Figure 3c. The compact representation can finally be expanded in order to enumerate the solutions.

Further Improvements As a first important improvement, we can derive that a CSP has no solution as soon as one of its independent partial problems has no solution. This is also reflected in multiplying the solution counts. A simple improvement is to skip counting in further partial problems, whenever a partial problem returns no solutions.

By this, the order of the partial problems is critical for avoiding unnecessary work. Optimally, partial problems with high chance of failing are explored first. The ordering can be based on heuristics analogous to the variable selection for branching. Additional savings result from checking if all partial problems are satisfiable, before we start to count all solution of any partial problem.

The solution number for partial problems with an empty set of constraints can be derived directly without further decomposition or enumeration. In this case, the number of solutions can be determined as product of the domain sizes. This effect is already shown in Figure 2.

DDFS profits most from early and well balanced decompositions. Therefore, new strategies for variable and value selection are desirable that support good decomposition. Constraint graph based variable selection, e.g. detection of articulation points, can guide the variable selection and domain splitting instead of single value branching may lead to sturdy decompositions.

Even deeper analysis of the constraint graph structure can guide the heuristics further. Many techniques are already investigated and proved beneficial for the static case [14, 15, 18, 23]. For example, we could strive for the breaking of circles in the constraint graph in order to obtain a tree structure. Solutions of a CSP with tree structure can be enumerated much more efficiently. Note that the detecting when the graph becomes a tree comes for free if we already look for graph decompositions.

Albeit presented in this fashion, DDFS is not completely restricted to binary constraint graphs. Many widely used n-ary and global constraints (e.g. `AllDifferent`) can be used as well, if a suitable binarization is at hand [5, 21]. The method can then employ the strong propagation of the global constraint and use the semantically equivalent set of binary constraints for checking dependencies in the constraint graph.

3 Application to Structure Prediction and Results

In [4], a constraint-based approach for exact structure prediction in the *HP-model* of the cubic and face-centered cubic lattice has been presented. In this simplified protein model, the amino-acids of the protein are classified into *hydrophobic (H)* and *polar (P)* ones and each is represented by a single point, its center of mass. A *structure* is a placing of these H/P-monomers to nodes of the lattice, such that successive monomers are lattice neighbors and each node is only occupied once (*self-avoidance*). The energy is calculated as shown in Figure 3a by counting HH-contacts. The example structure for the sequence HPPHPPHPHP in Figure 3b has an energy of -2.

a)

	H	P
H	-1	0
P	0	0

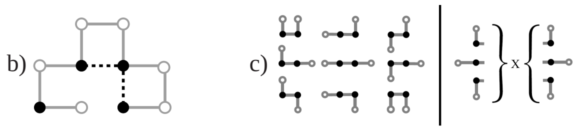

Fig. 3. a) Energy function b) structure of square lattice HP-model (H-monomer: black, P-monomer: white, structure back bone: grey, HH-contact: dotted) c) Structure space compression.

The prediction of *optimal* structures (with minimal energy) can be formulated as CSP and was named *Constraint-based Protein Structure Prediction (CPSP)* [2, 4]. It is a fast approach for enumerating all structures for a given HP-sequence using DFS.

Its main idea is the pre-calculation of so called *optimal H-cores*, a set of positions for a fixed number of H-monomers that minimizes the energy function. The construction of this sets is a hard problem by itself, which was solved using constraint-programming too [3, 24]. Since H-cores are sequence-independent, they can be pre-calculated and used for the last sequence-specific part of CPSP. For the remaining task, its necessary to search for self-avoiding walks with the restriction that H-monomers are placed in a given H-core. Therefore, we introduce a variable for each sequence position with lattice nodes as values. H-monomers are constrained to H-core positions, P-domains are left with a finite domain of non-H-core positions. The self-avoiding walk condition can be expressed by a global `AllDifferent` constraint and a sequence of neighbor constraints, which can be modeled as $X_i - X_{i+1} = N_i$. There, X_i represents the variable for the ith sequence position and N_i contains all possible lattice specific neighbor vectors[1].

CPSP is effectively solved using DFS and so CDFS for solution counting can be applied too. As mentioned before the number of optimal structures of a protein is an important measure. It provides information about the character of the energy landscape and the degeneracy and can be used for their further investigation [11, 10, 12, 20, 26, 27].

As discussed before a semantically equal set of binary inequality constraints can be used to represent the global `AllDifferent` constraint in the constraint graph. DDFS was applied using problem specific heuristics in addition to node degree and articulation point identification. A first prototypical implementation uses ILOG Solver 6.1$^{\text{TM}}$. We present some results from this program in the following table.

test suite	branch	fail	time	pos. time	decomp.
T33	7.8	0.7	1.5	4.7	42
T54	7.7	0.9	1.7	5.2	26

[1] In practice, lattice positions and the neighborhood N are indexed by integers such that standard constraint solvers for finite domains over integers are applicable.

We investigated two test suites T33/T54 with random HP-sequences of length 33/54 in the cubic lattice. To show the contraction of the search tree the ratio of branchings CDFS/DDFS is given in column *branch*. It can be reduced by decomposition up to a factor of 8 in average with the presented average number of decompositions. Due to a non-optimal partial problem ordering DDFS yields a little higher number of *fails* during the enumeration. The ratio of the mean time consumption of CDFS/DDFS in column *time* illustrates the reduced number of branching. This time-behavior can certainly be improved, since the current implementation is not at all optimized. The time ratio in column *pos. time* is calculated using DDFS without versus with decomposition. It demonstrates the possible speedup using DDFS for faster implementations and is around 5 times. We expect further speedups and search tree reductions using better partial problem ordering and variable selection heuristics.

4 Discussion

We presented a general method Decomposing DFS (DDFS) for completely counting and enumerating the solutions of a binary CSP by dynamically exploiting decomposition of (sub-)CSPs. Furthermore, we demonstrated that the method can be generalized such that even global constraints can be used. As we could show, our strategy of dynamically decomposing the (sub-)problems into partial problems reduces the search tree significantly. Since partial problems can be efficiently detected using well established graph algorithms, this results in a speed up of the search. Beyond this, we discussed how the graph structure can guide the variable and value selection in order to achieve many balanced decompositions, e.g. by the identification of articulation points. Such considerations go beyond previous work on constraint graph decomposition.

The application of DDFS to the CPSP problem shows the large capabilities of the method. First results with a prototypic implementation already show a significant speedup. Improving our ability for counting and enumerating optimal structures has important implications for the investigation of protein evolution and the folding process.

We could give evidence that the more general approach of dynamically analyzing the constraint graph during the search and employing its special structure has a large potential for solution counting in constraint programming. To our conviction, exploring these possibilities even further is an interesting field for future research.

Acknowledgments Martin Mann is supported by the EU project EMBIO (EC contract number 012835). Sebastian Will is partly supported by the EU Network of Excellence REWERSE (project reference number 506779).

References

1. Ola Angelsmark and Peter Jonsson. Improved algorithms for counting solutions in constraint satisfaction problems. In *Proc. of CP-2003)*, pages 81–95, Sep-Oct 2003.
2. Rolf Backofen and Sebastian Will. Fast, constraint-based threading of HP-sequences to hydrophobic cores. In *Proc. of CP'2001*, volume 2239, pages 494–508, 2001.
3. Rolf Backofen and Sebastian Will. Optimally compact finite sphere packings — hydrophobic cores in the FCC. In *Proc. of CPM2001*, pages 257–272, 2001.

4. Rolf Backofen and Sebastian Will. A constraint-based approach to fast and exact structure prediction in three-dimensional protein models. *J. of Constraints*, 11(1):5–30, 2006.
5. Roman Bartk. Theory and practise of constraint programming. In *CPDC2001*, pages 7–14. Wydavnictvo Pracovni Komputerowej, Gliwice, Poland, 2001.
6. B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *JCB*, 5(1):27–40, 1998.
7. Armin Biere and Carsten Sinz. Decomposing sat problems into connected components. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–198, 2006.
8. Elazar Birnbaum and Eliezer L. Lozinskii. The good old davis-putnam procedure helps counting models. *Journal of AI Research*, 10:457–477, 1999.
9. Rina Dechter and Robert Mateescu. The impact of AND/OR search spaces on constraint satisfaction and counting. In *CP'2004*, 2004.
10. Ken A. Dill and Hue S. Chan. From levinthal to pathways to funnels. *Nature Structural Biology*, 4(1):10–19, 1997.
11. A. R. Dinner, A. Sali, and M. Karplus. The folding mechanism of larger model proteins: Role of native structure. *Proc. Natl. Acad. Sci. USA*, 93:8356–8361, 1996.
12. Christoph Flamm, Ivo L. Hofacker, Peter F. Stadler, and Michael T. Wolfinger. Barrier trees of degenerate landscapes. *Z.Phys.Chem*, 216:155–173, 2002.
13. Eugene C. Freuder and Michael J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proc. of IJCAI-85*, pages 1076–1078, 1985.
14. G. Gottlob, N. Leone, and F.Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.
15. G. Gottlob, N. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.
16. Peter Grassberger. Sequential Monte Carlo methods for protein folding. In *NIC Symposium 2004*, Juelich, oct 2004.
17. R. J. Bayardo Jr. and J. D. Pehoushek. Counting models using connected components. In *Proc. of the 7th Nat'l Conf. on AI*, 2000.
18. Philippe Jgou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *LSIS*, 2002.
19. Gilles Pesant. Counting solutions of CSPs: A structural approach. In *IJCAI-05*, page 260, 2005.
20. A. Renner and E. Bornberg-Bauer. Exploring the fitness landscapes of lattice proteins. In *2nd. Pacif. Symp. Biocomp.* Singapore, 1997.
21. F. Rossi and V. Dhar. On the equivalenve of constraint satisfaction problems. In *ECAI90*, pages 550–556. Stockholm, Sweden, 1990.
22. Dan Roth. On the hardness of approximate reasoning. *Artif. Intelligence*, 82(1-2):273–302, 1996.
23. Marko Samer. Hypertree-decomposition via branch-decomposition. In *IJCAI'05*, pages 1535–1536, 2005.
24. Sebastian Will. Constraint-based hydrophobic core construction for protein structure prediction in the face-centered-cubic lattice. In *Proc. of PSB 2002*, pages 661–672, 2002.
25. M. Wolfinger, S. Will, I. Hofacker, R. Backofen, and P. Stadler. Exploring the lower part of discrete polymer model energy landscapes. *Europhysics Letters*, 74(4):725–732, 2006.
26. Richard Wroe, Erich Bornberg-Bauer, and Hue Sun Chan. Comparing folding codes in simple heteropolymer models of protein evolutionary landscape: robustness of the superfunnel paradigm. *Biophys J*, 88(1):118–31, 2005.
27. K. Yue, K. M. Fiebig, P. D. Thomas, H. S. Chan, E. I. Shakhnovich, and K. A. Dill. A test of lattice protein folding algorithms. *PNAS*, 92(1):325–9, 1995.